[redacted]

#hojas
7

Ej.1	Ej.2	Ej.3	Ej.4	Nota
B-	B	B	B-	A

Corrector: FRAN

Organización del Computador I - **Parcial**

Segundo Cuatrimestre 2016

Aclaraciones

- Anoté apellido, nombre, LU y numere *todas* las hojas entregadas
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente.
- El parcial **NO** es a libro abierto, pero puede tener los apuntes provistos por la cátedra y dos hojas A4 con apuntes propios.
- **Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluídas en el examen.
- El parcial se aprueba con los ejercicios 1, 2 Bien y del 3 y 4 al menos uno Regular.

Ejercicio 1 Se desea realizar el diseño de la MiniRobot. Esta máquina tendrá arquitectura Von Neumman, un *Program Counter* de 6 *bits* y una memoria direccionable a palabra de 10 *bits*. Tendrá también 4 registros de proposito general de 10 *bits* y un registro especial llamado *BAT*. Contará con diferentes sensores: Ultrasonido para detectar objetos, encoders para ver cuánto avanzó cada rueda, sensor de batería; y dos motores. Tanto los sensores, como los motores serán accedidos desde direcciones de memoria reservadas, pero no fijas. Los motores sólo podrán moverse con 4 valores diferentes de cantidad de pasos. El set de instrucciones será el siguiente:

Instrucción	Efecto
`MOV_ULTRASONIDO Reg, MemDir`	`Reg` ← `[MemDir]`
`MOV_ENCODER Reg, MemDir`	`Reg` ← `[MemDir]`
`MOVER_MOTOR #Pasos, MemDir`	`[MemDir]` ← `Ext10bits(#Pasos)`
`VER_ESTADO_BAT MemDir`	`BAT` ← `[MemDir]`
`ADD RegA, RegB`	`RegA` ← `RegA + RegB`
`SUB RegA, RegB`	`RegA` ← `RegA - RegB`
`JMP MemDir`	`PC` ← `MemDir`
`JE MemDir`	`Si Z=1, PC` ← `MemDir`
`SUSPENDER_LOW_BAT`	`Si BAT<16, [0x3F]` ← `0x3FF`
`IRET`	`Vuelve de RAI`

a. Proponer un Formato de Instrucción de longitud fija para la MiniRobot.

b. Definir:

  I. El tamaño máximo de la memoria

  II. La cantidad de instrucciones sin operandos que podrían agregarse a este formato de instrucción.

  III. Se desea agregar la posibilidad, sin modificar el set de instrucciones provisto, para agregar la funcionalidad de **AVANZAR_ROBOT N, MemDirI, MemDirD**, que deberá tener el efecto de que el robot se desplace N posiciones, utilizando las direcciones de memoria de los motores Izquierdo y Derecho. Decidir si es posible o no. En caso afirmativo, indicar cómo.

Ejercicio 2

a. Realice el diagrama del *datapath* de una microarquitectura para la MiniRobot que soporte la ejecución de las instrucciones descriptas. Recuerde indicar el tamaño de cada registro, de los buses internos y externos, las señales de cada componente y justificar la utilización de cada componente escogido y cada decisión tomada.

Para realizar el *datapath* puede utilizar los siguientes componentes:

- una única ALU que realiza las operaciones ADD y SUB.
- un único incrementador que suma 1.
- un único extensor de signo.
- un único controlador de memoria.

Al incluirlos detallar cuidadosamente el tamaño de los registros y los nombres de las señales. Cualquier otro componente a utilizar deberá ser implementado e incluido en el examen.

b. Desarrolle un componente extra que tenga una entrada de 10 *bits*, una salida de 1 *bit* y que permita determinar si el estado de la batería (que tiene un rango de 0 a 1023), es menor a 16.

c. Escriba las microinstrucciones que debe ejecutar la máquina para realizar el *fetch* de una instrucción (no incluir etapas posteriores del ciclo de instrucción).

d. Escriba el microprograma que realiza la parte de ejecución del ciclo de instrucción de las siguientes instrucciones:

   I. `JE 0x1F`

   II. `MOVER_MOTOR 2,[0x34]` //Se asume que hay un motor mapeado en esa dirección.

   III. `SUSP_LOW_BAT` //Si el registro BAT es menor que 16, pone todo en 1 el registro de control de la batería, mapeado en la dirección 0x3F.

Ejercicio 3 El diseño de la MiniRobot ha sido un éxito y ya se cuenta con un prototipo. Se requiere programar algunos tests. Para ello, primero será necesario definir el mapeo de los sensores y motores en las últimas 16 direcciones de la memoria, reservadas para ello.

a. Realizar el mapeo para los siguientes dispositivos de E/S: Un sensor ultrasonido que usará un registro llamado US_STAT, dos motores con sus registros MOTI_CTRL y MOTD_CTRL, dos encoders con sus registros ENI_STAT y END_STAT; y un controlador de batería con sus dos registros BAT_STAT y BAT_CTRL (que deberá ser mapeado en la última dirección reservada).

b. Realizar un diagrama de interconexión del sistema indicando la dirección de los buses, las líneas implicadas y la ubicación de los registros de E/S.

c. Realizar un programa que primero chequee el estado de la batería y en caso de estar en nivel bajo, deberá suspender el robot. En caso contrario, el robot deberá avanzar siempre y cuando los dos encoders se mantengan iguales. Si esto no ocurre, el robot no sigue avanzando. Escribir un Pseudocódigo y programarlo con el Set de Instrucciones de la MiniRobot.

d. Se introduce una funcionalidad en el controlador de batería, que envía una interupcción cuando el nivel de la batería está bajo. Modificar el código anterior y, de ser necesario, el diagrama de interconexión para que la frecuencia de muestreo sobre cada encoder aumente. Comparar ambas variantes, dar ambas frecuencias de muestreo y explicar cuál esquema es mejor. Considere para este último análisis que el tiempo de ejecución de una instrucción que involucra E/S es de 2 ms, mientras que el resto de las operaciones demora 1 ms. Asuma además que los encoders arrojan el mismo valor.

Ejercicio 4 Se tiene el siguiente programa ASSEMBLY de una máquina Orga1:

```
data:  DW 0x0002
start: MOV R0, [R1+0x0003]
loop:  SUB R0, 0x0001
       JNE loop
end:   JMP end
```

Se pide:

a. Si el *PC*, cuyo valor es $0xA000$, referencia a la instrucción etiquetada por *start*, deducir los valores de *data*, *loop* y *end*.

b. Muestre el *dump* de memoria de dicho programa y calcule los desplazamientos para los saltos relativos.

c. Realice un diagrama de tiempos para describir el comportamiento del BUS al comunicarse el CPU con la memoria, tal como lo hace en la línea *start*.

d. Indicar la capacidad del BUS en relación al protocolo descripto en el punto anterior, asumiendo un ciclo de reloj de 100 kHz.

[redacted]

① MINIROBOT

- PC: 6 bits
- Direcc. a palabra de 10 bits.
- RØ, ..., R3 reg. de prop. general: 10 bits
- Registro especial BAT
- Sensores y motores tienen direcciones reservadas, pero no fijas.

Como el PC almacena una dirección de la memoria, sabemos que las direcciones son de 6 bits.
Se necesitan 2 bits para diferenciar 4 registros (ØØ-11).
Se necesitan 2 bits para diferenciar las 4 cantidades de pasos que puede dar el motor.

A continuación se grafica el espacio disponible para cada código de operación, que luego se decidirá. Como las palabras son de 10 bits, y todas las instrucciones entran en 10 bits, el formato de instrucción elegido es de 10 bits (fijos).

9	8	7	6	5	4	3	2	1	Ø	10 bits
C.O		REG		Mem. Dir						MOV-ULT...
C.O.		REG		Mem. Dir						MOV-ENC...
C.O.		#PASOS		Mem. Dir						MOVER-MOT...
C.O.				Mem. Dir						VER-ESTA...
C.O						RA		RB		ADD...

9	8	7	6	5	4	3	2	1	Ø	
C.O.						RA		RB		SUB...
C.O.				Mem. Dir						JMP...
C.O.				Mem. Dir						JE
C.O.										SUSPEND...
C.O.										IRET

Como son 10 instrucciones, no podemos minimizar el largo del C.O. al menor (2 bits) porque no llegamos a diferenciarlas. Luego, el cód. de op. variará en longitud.

Instrucción	Codificación									
MOV_VLT... RG, MD	Ø	Ø	R		M	M	M	M	M	M
MOV-ENC... RG, MD	Ø	1	R	R	M	M	M	M	M	M
MOVER-MOT #P, MD	1	Ø	P	P	M	M	M	M	M	M
VER-EST.BAT MD	1	1	Ø	Ø	M	M	M	M	M	M
JMP MD	1	1	Ø	1	M	M	M	M	M	M
JE MD	1	1	1	Ø	M	M	M	M	M	M
ADD RA, RB	1	1	1	1	Ø	Ø	RA	RA	RB	RB
SUB RA, RB	1	1	1	1	Ø	1	RA	RA	RB	RB
SUSPENDER-LOOO.B	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø
IRET	1	1	1	1	1	1	Ø	Ø	Ø	Ø
	9	8	7	6	5	4	3	2	1	Ø

Referencias:

- Los bits pintados son cód. de operación.
- Las "M" contiguas son la direcc. de memoria.
- Las "R", ó "RA" ó "RB" contiguas son los números de registros.
- Las "P" contiguas son los pasos del motor.

[redacted]

b)

i) Como la memoria es direccionable a 10 bits, y solo podemos diferenciar 2^6 direcciones con 6 bits, como máximo será de:

$$2^6 . 10 \text{ bits} = 640 \text{ bits} = 80 \text{ bytes}.$$

ii) Para que una nueva instrucción se diferencie de las existentes debe comenzar con:

"1 1 1 1 1 ∅ x x x x" ó "1 1 1 1 1 1 x x x x"

Sin embargo, en ningún caso pueden terminar con "... ∅∅∅∅", pues serán iguales a dos ya existentes.

Luego, las instrucciones disponibles son tantas como:

$$2 . \#\{∅∅∅1; .. ; 1111\} = 2 . 15 = \boxed{3∅.}$$

iii) Como la instrucción involucra dos direcciones de memoria (6 bits cada una), sabemos que la instrucción medirá más de $6 . 2 = 12$ bits.

Como ya se decidió que las instrucciones son de 10 bits de longitud fija, no sería posible agregar tal instrucción.

Una solución sería que la longitud de las instrucciones sea variable, y "AVANZAR-ROBOT N, MDI, MDD" debería requerir más de una palabra (varios fetch), pero NO es el caso.

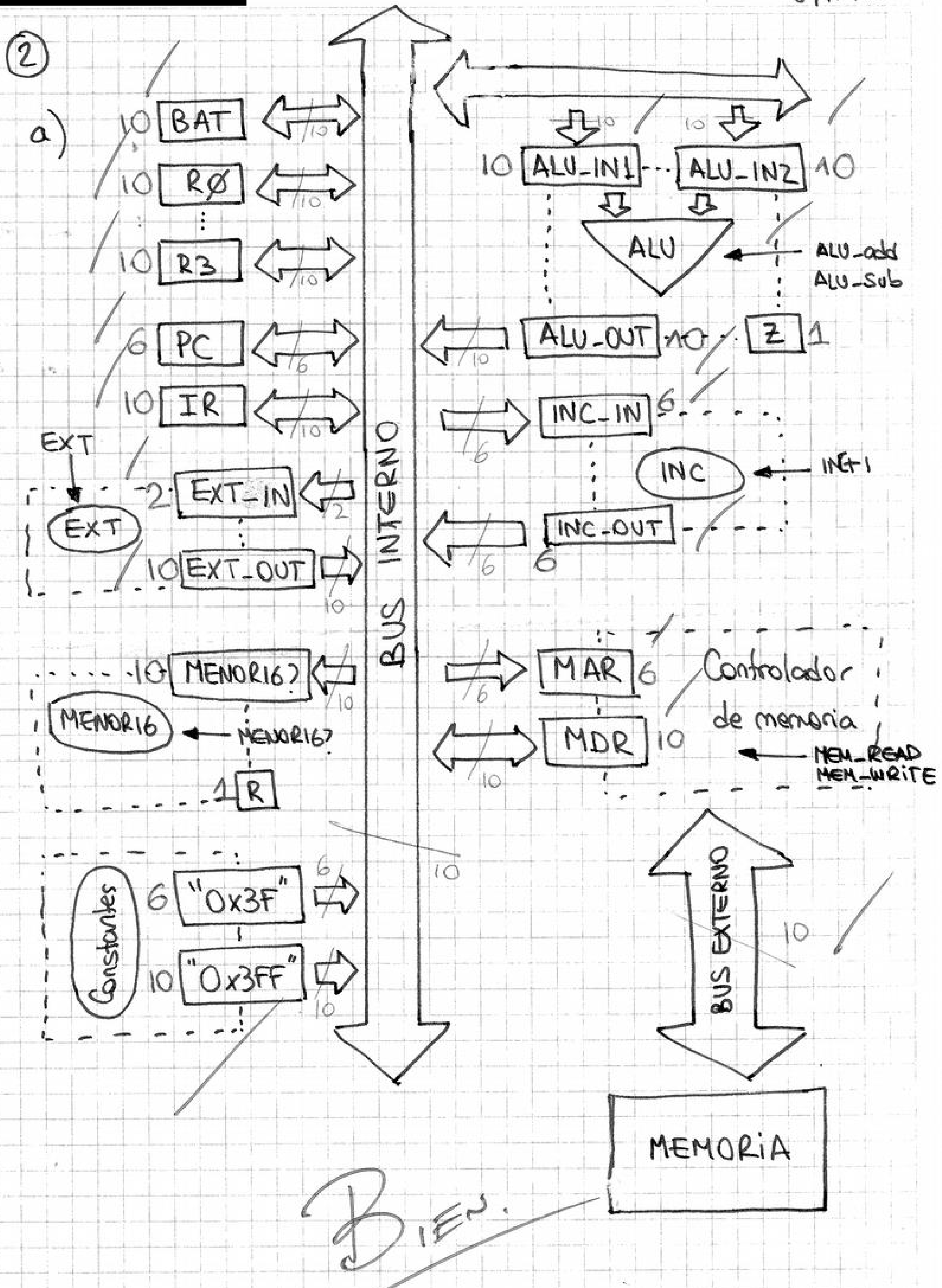
~~NO~~

PODRÍA EMULARSE CON OTRAS.

[redacted]

2

a)



Bien.

NOTA

Componentes utilizados:

- Constantes.
- menor16 (a implementar en ej. b)

Constantes

Bien.

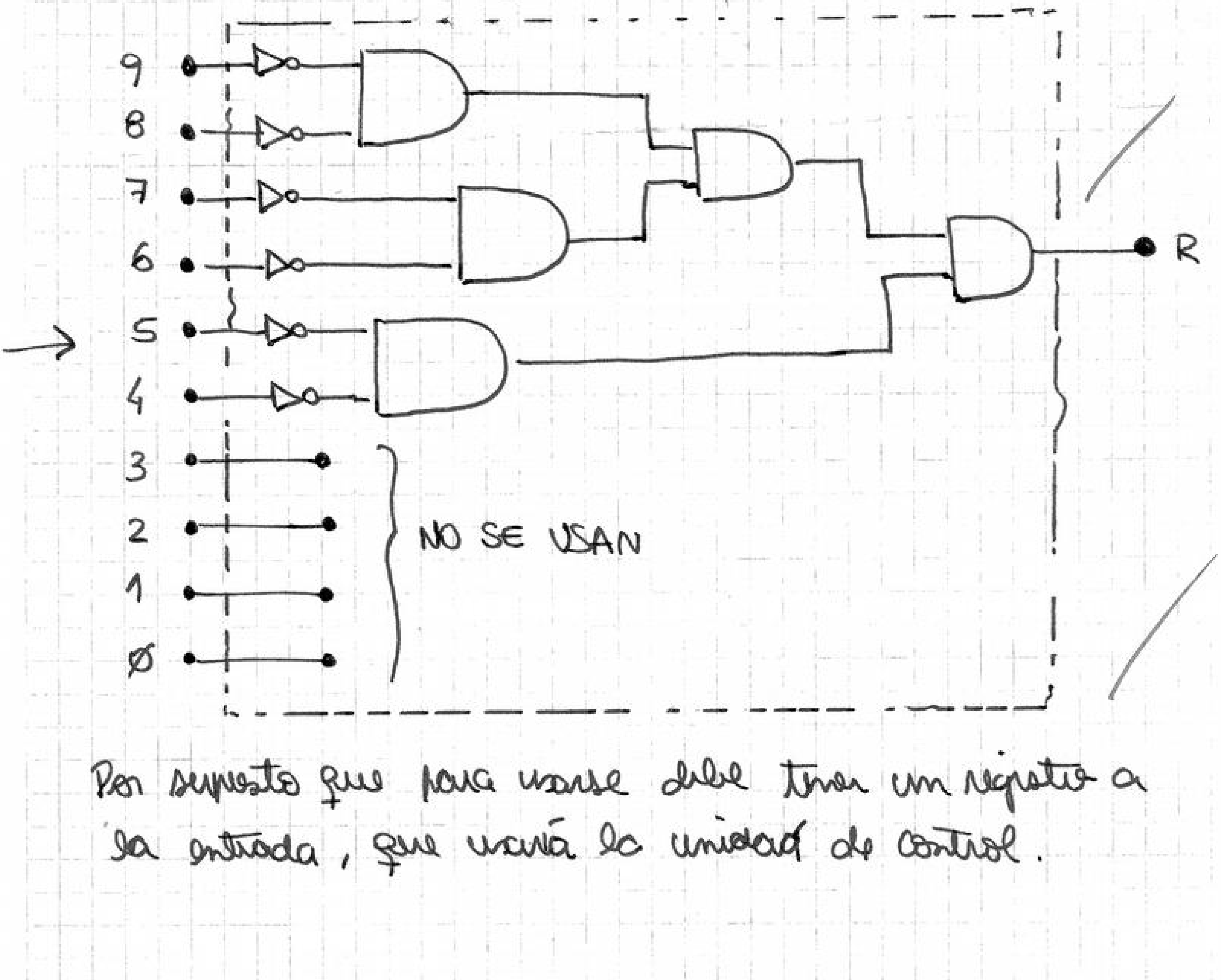
Notas sobre el datapath:

- El componente "constantes" es para poder implementar "SUSPENDER_LOW-BAT". Los registros son estáticos, siempre legibles.
- Los datos manejados son de 10 bits, las direcciones de 6 bits. Un caso particular es el de los pasos, que son de 2 bits, por lo que se necesita un extensor de signo para moverlo a la memoria.
- MAR y MDR son los registros de direcciones y datos respectivamente del controlador de la mem. Es por eso que son de 6 y 10 bits.
- Sólo será necesario el flag "Z" de la ALU, por las instrucciones a implementar.
- El comp. "menor16" se utilizará para chequear si BAT<16 en la ejecución de "SUSPENDER_LOW-BAT".

[redacted]

- También debería haber un registro de 1 bit que indique si se puede interrumpir al CPU; no figura en el datapath.
- El incrementador se utilizará para aumentar el PC.

b) Como es un número en codificación sin signo, será menor a 16 cuando el bit 4 y todos los más significativos que este sean cero.
Los bits ∅, 1, 2 y 3 no aportan información.



Por supuesto que para usarse debe tener un registro a la entrada, que usará la unidad de control.

c) Fetch

MAR := PC

• MEM_READ

IR := MDR

INC_IN := PC

• INC+1

PC := INC-OUT

d) i) JE 0x1F

if z=1 {

PC := IR [5...0]

}

ii) MOVER_MOTOR 2, [0x34]

MAR := IR [5...0]

EXT_IN := IR [7...6]

• EXT

MDR := EXT_OUT

MEM_WRITE

iii) SUSP_LOW-BAT

MENOR16? := BAT

• MENOR16?

if R=1 {

MAR := "0x3F"

MDR := "0x3FF"

MEM_WRITE

}

¡Es el nombre del registro, no el número!

BIEN.

[redacted]

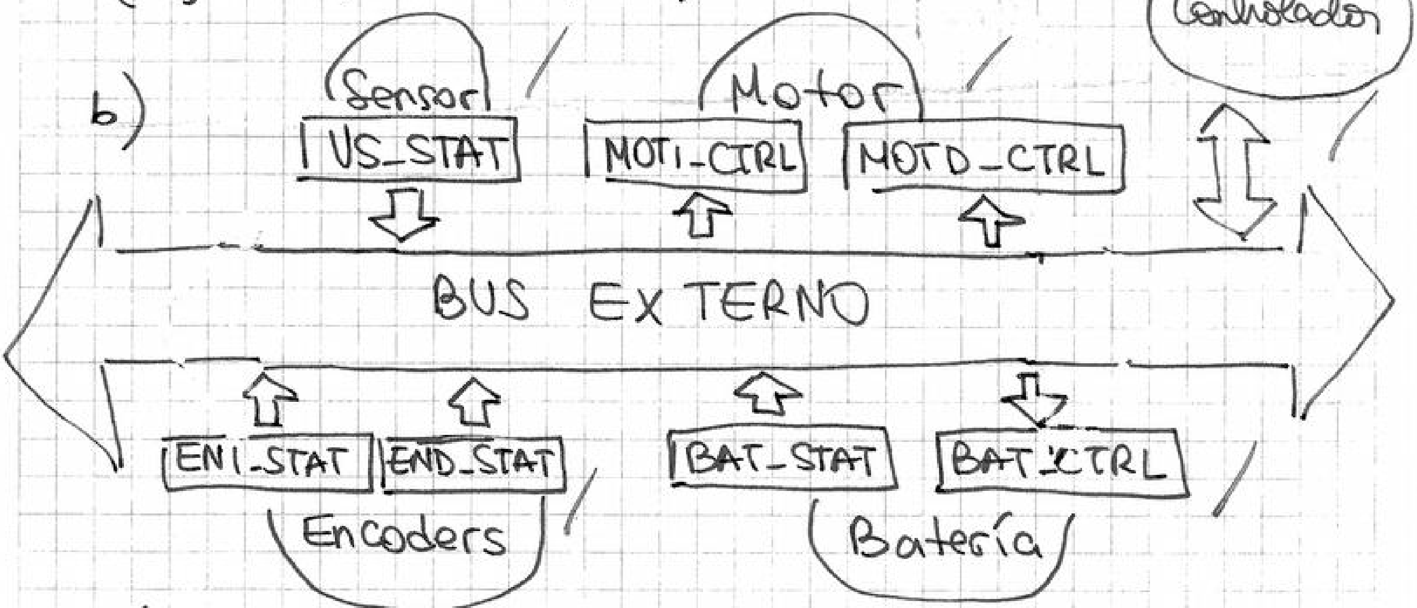
③

a)

US_STAT ⟼ 0x39
MOTI_CTRL ⟻ 0x3A
MOTD_CTRL ⟻ 0x3B
ENI_STAT ⟼ 0x3C
END_STAT ⟼ 0x3D
BAT_STAT ⟼ 0x3E
BAT_CTRL ⟻ 0x3F

Las flechas indican el sentido del flujo de información (registro dispositivo – memoria).

b)



Los dispositivos están conectados al BUS Externo, y el CPU se comunica con ellos por medio del controlador de memoria.

c)
- Actualizar registro de batería.
- Si es bajo, suspender.
- Si no:
  Si ENI = END, avanzar.
- Repetir

```
ciclo: VER_ESTADO_BAT   0x3E // 2 ms
       SUSPENDER_LOW_BAT // 2 ms
       MOV_ENCODER RØ, 0x3D // 2 ms
       MOV_ENCODER R1, 0x3C // 2 ms
       SUB R1, RØ // 1 ms
       JE iguales // 1 ms
       JMP ciclo
iguales: MOVER_MOTOR Ø1, 0x3A // 2 ms
         MOVER_MOTOR Ø1, 0x3B // 2 ms
         JMP ciclo // 1 ms
```

(Esto es suponiendo que al poner el control de la batería en 1111111111 se suspende el robot).

d)

```
ciclo: MOV_ENCODER RØ, 0x3D // 2ms
       MOV_ENCODER R1, 0x3C // 2ms
       SUB R1, RØ // 1 ms
       JE iguales // 1 ms
       JMP ciclo
iguales: MOVER_MOTOR Ø1, Ø x3A // 2ms
         MOVER_MOTOR Ø1, 0x3B // 2ms
         JMP ciclo // 1 ms
```

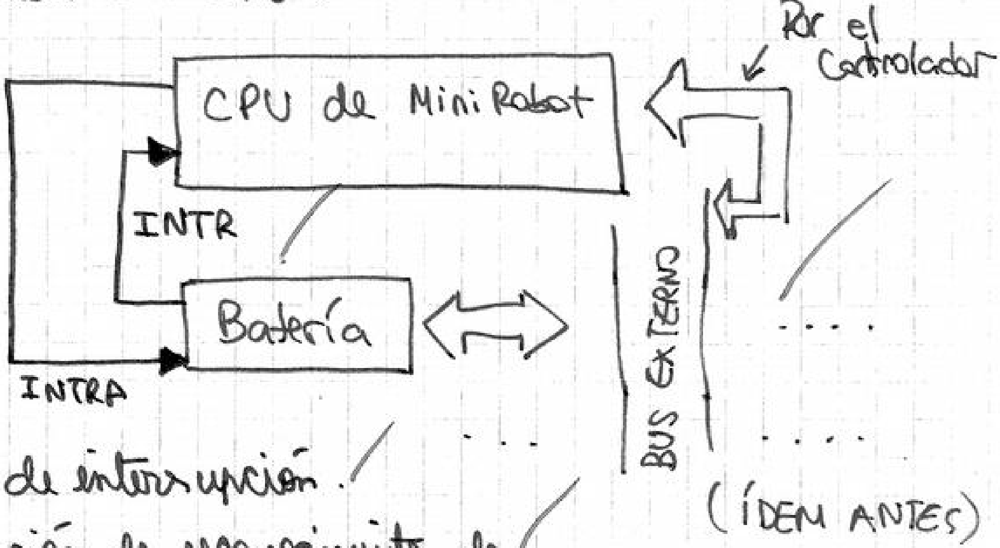
interrupción: VER_ESTADO_BAT 0x3E
SUSPENDER_LOW_BAT
~~IRET~~.

NO

Bien

[redacted]

Las conexiones al BUS externo serán las mismas. Sin embargo, ahora la batería tendrá su señal de interrupción conectada directo al CPU:



- INTR: línea de interrupción.
- INTRA: indicación de reconocimiento de interrupción.

Antes un ciclo tardaba: $2ms \times 6 + 1ms \times 3 = 15ms$.
Con la interrupción tarda: $2ms \times 4 + 3 \times 1ms = 11\ ms$

Como se toma una muestra del encoder por ciclo, las frecuencias de muestreo son:

$T = 15\ ms \rightsquigarrow \boxed{f = 66,\bar{6}\ Hz}$ ← SIN INTERRUPCIÓN

$T = 11 ms \rightsquigarrow \boxed{f = 90,\overline{90}\ Hz}$ ← CON INTERRUPCIÓN

Es claro que es mejor tener una interrupción en vez de hacer polling constantemente. Esto permite que otros polling se puedan hacer con más frecuencia (los de los encoders).

[redacted]

FECHA 08/11/16

(4)

	# pal.	pos. mem.
data : DW 0x0002	1	0x9FFF
start : MOV RØ, [R1 + 0x0003]	2	0xA000
loop : SUB RØ, 0x0001	2	0xA002
JNE loop	1	0xA004
end : JMP end	2	0xA005

a)

data ≡ 0x9FFF

start ≡ 0xA000

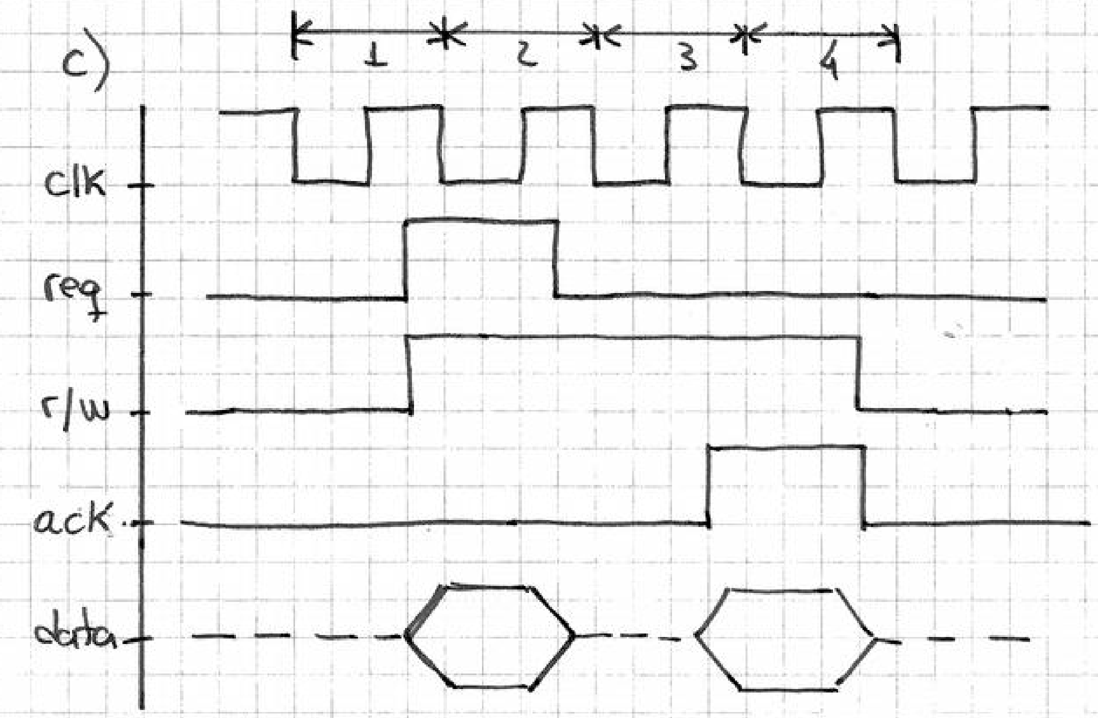
loop ≡ 0xA002 (⇝ JNE loop ≡ JNE 0xF~~A~~)
D

end ≡ 0xA005

Bien.

b) (Ya lo dice en el punto ~~anterior~~) Falta DUMP.
No

c)



Bien.

Aquí se comparten las líneas de data para ~~.........~~ direcciones y para datos, y se comparten entre dispositivos. Es por eso que se liberan "- - -" ≡ HiZ.

NOTA

d) El protocolo descripto trajo una palabra de 16 bits (es la Orga 1) en 4 ciclos (recordemos que medimos el flujo de datos que consideramos información, y no así las direcciones).

Si cada ciclo tarda $\frac{1}{100KHz} = \frac{1}{100000\,Hz} = \frac{1}{100000}$ seg $= 1.10^{-5}$ segundos,

entonces el BUS mueve $\frac{16}{4.1.10^{-5}}$ bits/seg $= 400000$ bits/seg

400000 bits/seg = 50000 bytes/seg = $\boxed{50\ KB/seg}$

YAPA:

Pequeña descripción del protocolo de lectura propuesto:

- El CPU levanta req y pone la dirección que busca en data durante un ciclo.
- En ese ciclo la memoria detecta el req y la indicación de lectura, y lee la dirección de data. Luego espera a que se liberen las líneas de data para poner el dato y levantar ack (por un ciclo).
- El CPU lee los datos de data, y baja la señal de lectura. FIN.

Muy Buen Parcial

NOTA